Nathaniel Serrano

COS 420

Program 8

4.16.2025

Lab Book

Time spent working with AI: April 15, 8:30 pm - 10:00 pm

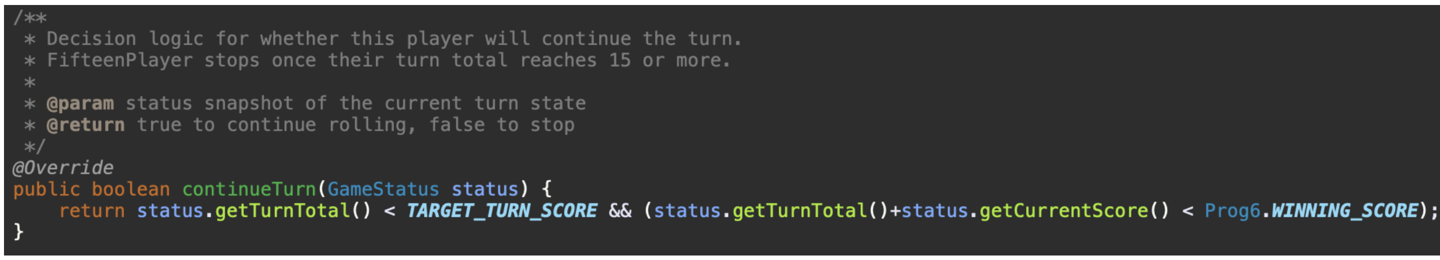
I began this assignment by having ChatGPT 4o create a class called GameStatus to encapsulate the status of the bulldog game. This was pretty simple and only required the following prompt: “We are going to rework how the player classes work since they have some duplicated code between them. We should take the code they all have in common and place it in Player.java. The unique parts of their respective play() methods should be put into methods called continue(). To begin, let's start by creating a class to encapsulate the status of the game. This should be a relatively simple class.” This prompt allowed me to have the AI tool produce GameStatus.java while also preparing it for refactoring the abstract and concrete Player classes. Next, I had it refactor the Player class, which led to it moving the bulk of the play() method there and creating an abstract method called continueTurn() that all child classes must fulfill.

To test that these changes worked, I had the AI tool create a continueTurn() method for FifteenPlayer, one of the simpler Player classes. I had to manually add the condition that the FifteenPlayer stops once their score is above the winning score threshold, but after that it was complete. Next I had the AI tool create the continueTurn() methods for other classes in the following order: RandomPlayer, WimpPlayer, UniquePlayer, NathanielUniquePlayer, SevenPlayer, and HumanPlayer. In hindsight, I should have done SevenPlayer immediately after FifteenPlayer since they were so similar (in fact, the new methods are exactly the same–more on that later). That being said, I saved HumanPlayer for last which was a good idea since I had forgotten that I had removed its CLI functionality to ensure it worked with the Bulldog GUI. This meant that when I asked the AI tool to create a continueTurn() method, it had it just return false since the actual logic was handled in the GUI class. To fix this, I asked Chat GPT 4o: “what if we run Prog6.java instead of BulldogGameGUI.java and want to use the HumanPlayer?” The AI tool then explained the issue of how HumanPlayer currently does not function in a console-based environment, and rewrote the continueTurn() method so that it only worked in a CLI, but offered the possibility of injecting a flag into HumanPlayer or creating subclasses like GuiHumanPlayer and CliHumanPlayer. I asked ChatGPT 4o to inject a flag into HumanPlayer, and it did just that, leading to what is shown below in its code snippet. This meant some minor changes to Prog6.java and BulldogGameGUI.java consisting of adding the parameters “true” and “false” respectively to newly constructed HumanPlayer objects to represent if HumanPlayer was being used in a console environment. This was found to work without issue.

I found that creating the continueTurn() method for all other classes before HumanPlayer was the better move, since HumanPlayer definitely required the most thought out of all of them. If I were to repeat this assignment, I would probably change up the order in which I implemented continueTurn() in some classes, like how I mentioned with SevenPlayer and FifteenPlayer. Outside of that, the general sequence of tasks accomplished in this assignment worked well and did not take long to complete, since each step fluidly transitioned to the next.

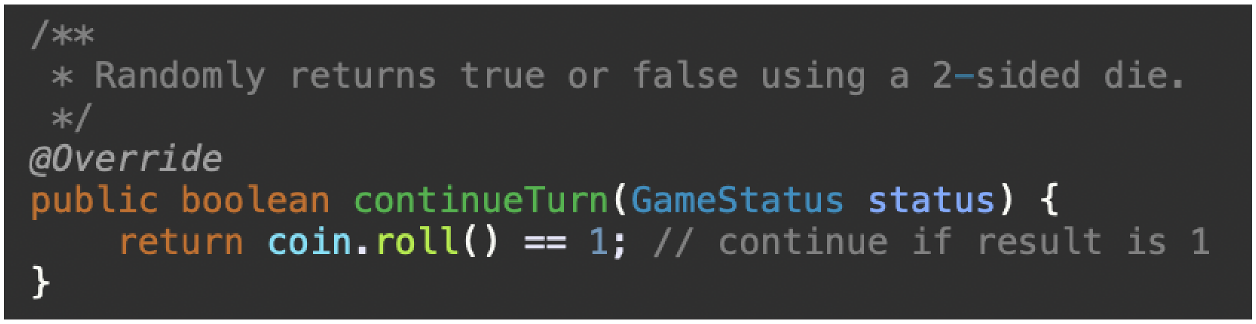
Code Snippets of each class's continueTurn() method

FifteenPlayer:

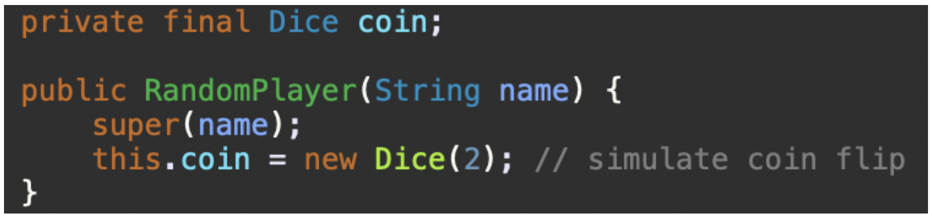


```java
/**
 * Decision logic for whether this player will continue the turn.
 * FifteenPlayer stops once their turn total reaches 15 or more.
 *
 * @param status snapshot of the current turn state
 * @return true to continue rolling, false to stop
 */
@Override
public boolean continueTurn(GameStatus status) {
    return status.getTurnTotal() < TARGET_TURN_SCORE && (status.getTurnTotal()+status.getCurrentScore() < Prog6.WINNING_SCORE);
}
```

RandomPlayer:

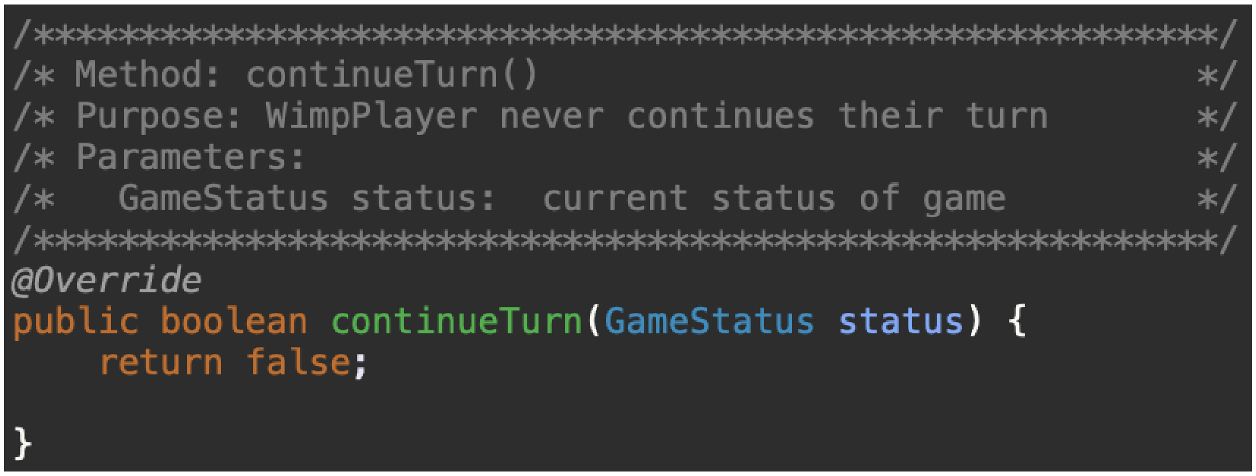


```java
/**
 * Randomly returns true or false using a 2-sided die.
 */
@Override
public boolean continueTurn(GameStatus status) {
    return coin.roll() == 1; // continue if result is 1
}
```

- Note: coin is a RandomDice object created in the constructor with 2 sides
    - 

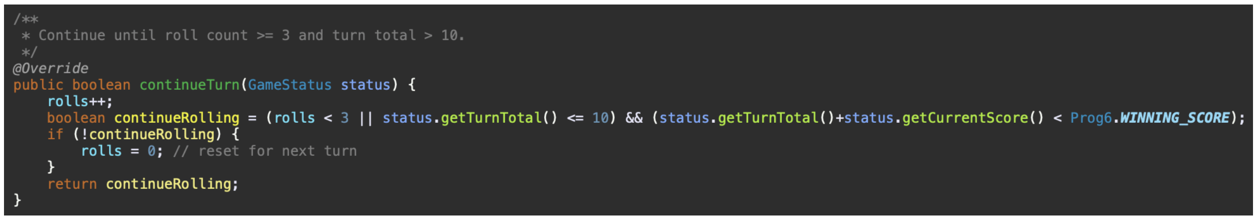
```java
private final Dice coin;

public RandomPlayer(String name) {
    super(name);
    this.coin = new Dice(2); // simulate coin flip
}
```

WimpPlayer:



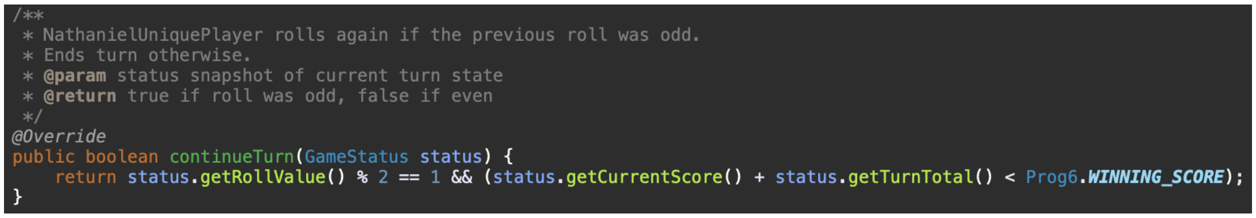
```
/**********************************************************/
/* Method: continueTurn()                                 */
/* Purpose: WimpPlayer never continues their turn         */
/* Parameters:                                            */
/*   GameStatus status:  current status of game           */
/**********************************************************/
@Override
public boolean continueTurn(GameStatus status) {
    return false;

}
```

UniquePlayer:



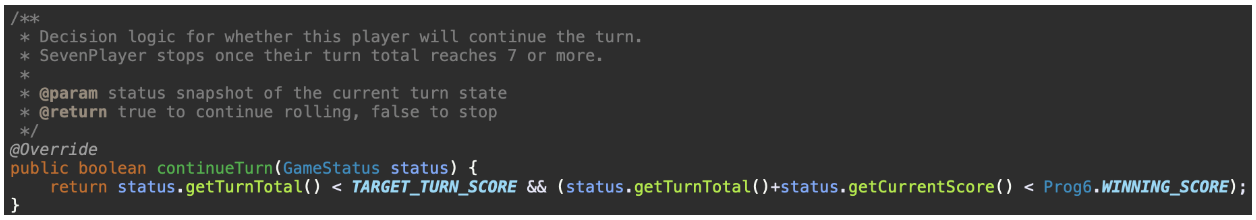
```
/**
 * Continue until roll count >= 3 and turn total > 10.
 */
@Override
public boolean continueTurn(GameStatus status) {
    rolls++;
    boolean continueRolling = (rolls < 3 || status.getTurnTotal() <= 10) && (status.getTurnTotal()+status.getCurrentScore() < Prog6.WINNING_SCORE);
    if (!continueRolling) {
        rolls = 0; // reset for next turn
    }
    return continueRolling;
}
```

NathanielUniquePlayer:



```
/**
 * NathanielUniquePlayer rolls again if the previous roll was odd.
 * Ends turn otherwise.
 * @param status snapshot of current turn state
 * @return true if roll was odd, false if even
 */
@Override
public boolean continueTurn(GameStatus status) {
    return status.getRollValue() % 2 == 1 && (status.getCurrentScore() + status.getTurnTotal() < Prog6.WINNING_SCORE);
}
```

SevenPlayer:



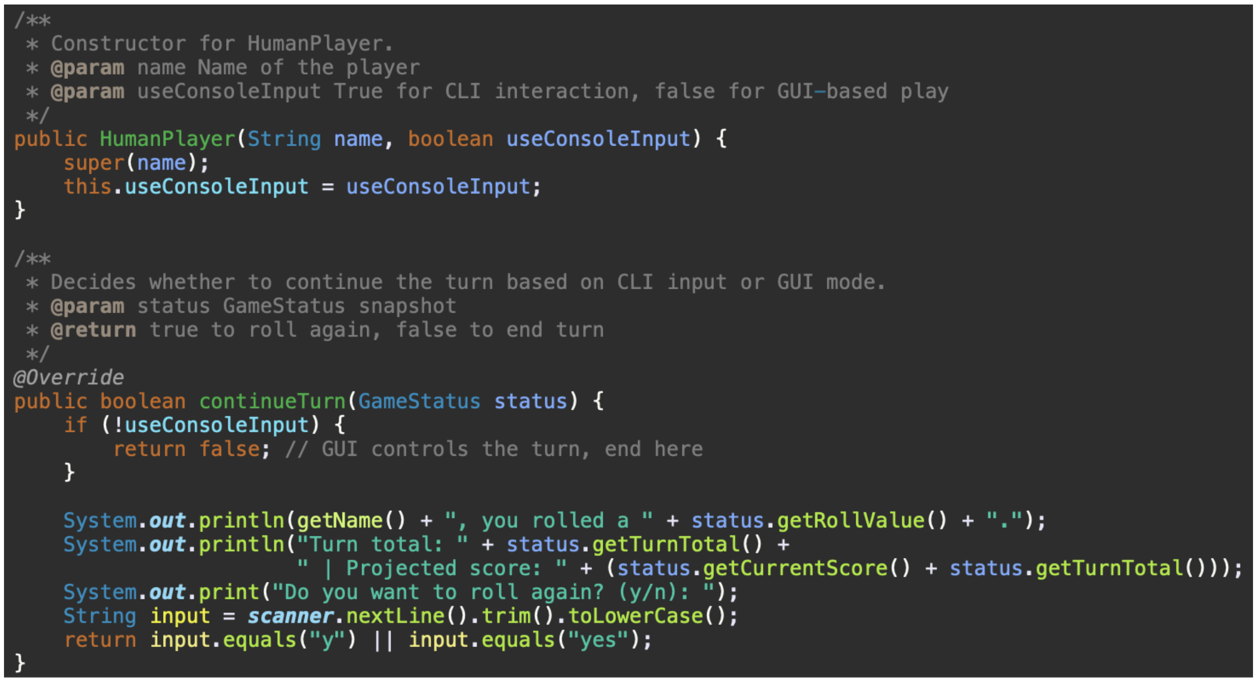
```
/**
 * Decision logic for whether this player will continue the turn.
 * SevenPlayer stops once their turn total reaches 7 or more.
 *
 * @param status snapshot of the current turn state
 * @return true to continue rolling, false to stop
 */
@Override
public boolean continueTurn(GameStatus status) {
    return status.getTurnTotal() < TARGET_TURN_SCORE && (status.getTurnTotal()+status.getCurrentScore() < Prog6.WINNING_SCORE);
}
```

- Note: SevenPlayer and FifteenPlayer’s continueTurn methods are the same, meaning we have some duplicated code here (only value that’s different is the

TARGET_TURN_SCORE). Typically, I would combine them into some TargetScorePlayer class, but chose to keep them separate here since SevenPlayer is still using FakeRandom from Program 7. Once we are finished with all Bulldog assignments, I plan to create this TargetScorePlayer class in my Bulldog GitHub repository.

HumanPlayer:

- HumanPlayer required a slightly different approach from the others to ensure it worked in both the GUI environment and the CLI. To address this, the AI tool added a flag to the HumanPlayer constructor, allowing for different return values in continueTurn() depending on the flag.



```java
/**
 * Constructor for HumanPlayer.
 * @param name Name of the player
 * @param useConsoleInput True for CLI interaction, false for GUI-based play
 */
public HumanPlayer(String name, boolean useConsoleInput) {
    super(name);
    this.useConsoleInput = useConsoleInput;
}

/**
 * Decides whether to continue the turn based on CLI input or GUI mode.
 * @param status GameStatus snapshot
 * @return true to roll again, false to end turn
 */
@Override
public boolean continueTurn(GameStatus status) {
    if (!useConsoleInput) {
        return false; // GUI controls the turn, end here
    }

    System.out.println(getName() + ", you rolled a " + status.getRollValue() + ".");
    System.out.println("Turn total: " + status.getTurnTotal() +
                       " | Projected score: " + (status.getCurrentScore() + status.getTurnTotal()));
    System.out.print("Do you want to roll again? (y/n): ");
    String input = scanner.nextLine().trim().toLowerCase();
    return input.equals("y") || input.equals("yes");
}
```